grepo - Repository group support

Srini, rs@toprllc.com

June 1, 2020

1 Introduction

Projects with many components and developers are best managed with distinct independent source code repositories. At a high level, there are the public repositories developed outside the team which are just used; the components that are unique to the project then may be viewed as private.

The public repositories then are primarily replicated for usage, prepared (eg. built) and periodically refreshed. The private repositories on the other hand go through an active development cycle. In git parlance, in addition to normal software engineering activities would include creating new branches, merging, tagging and of course publishing. Specific sub projects thus may well span several repositories.

This tool enables developers manage their **projectlets** as a group. For example create a new **feature branch** on a set of related repositories and finally push or publish the branch as a group once the developer is ready to share.

`https://source.android.com/setup/develop/repo` is such a tool which serves as a pattern for this projectlet. The goals are simplified somewhat:

- **git** is the only source code control system supported.
- Publishing updates, tags etc are to the original source repositories. The complete link to the repositories and the histories is maintained. The individual repositories may be manipulated individually.
- Commands mirror their **git** equivalents somewhat. The command semantics are designed for convenience and not for power. If necessary git can be directly accessed for more complex operations.
- Project configuration is provided in **yaml** format.

1.1 Typical use cases

Developers of embedded systems using e.g. **yocto** have to manage their environment as outlined above. Their own application might be split into several repositories while depending heavily on the platform libraries spread over numerous other repositories. Both these groups are evolving - the public ones presumably changing slower than the private ones.

Even simpler, many hobbyists have quite a few projectlets going in parallel, related somewhat loosely. In my own case my repos are spread between gitlab, github, bitbucket and so on. I also jump around from a Windows laptop, a Macbook and quite a few Linux servers not to mention several Raspberry PI's and I want to have most of my projects available in each of my systems. This tool allows me to setup my environment with minimal fuss.

2 Specifications

2.1 Configuration

A sample configuration for a project file is listed below.

Listing 1: Example Project.yaml

```
public:
  workarea: "$HOME/grepotest/pub"
  projects:
    - repourl: "git@gitlab.com:projtemplates/go.git"
      reference: "master"
      path: "go/lib"
      build: "make setup all"

private:
  workarea: "$HOME/grepotest/pvt"
  server: "git@gitlab.com:RajaSrinivasan/"

  projects:
    - repo: "random.git"
      path: "random"

    - repo:  "exec.git"
      path: "exec"
      build: "build.sh"

    - repo: "codex.git"
      path: "codex"
      reference: "v0.1.0-B"
      build: "make setup all test"

    - repourl: "https://github.com/RajaSrinivasan/disign.git"
      path: "disign"
      build: "make all"

    - repourl: "https://github.com/RajaSrinivasan/spm.git"
      path: "spm"
      build: "make all"

    - repourl: "https://github.com/RajaSrinivasan/srctrace.git"
      path: "srctrace"
      reference: "feat_many_platforms"
      build: "make setup all"

    - repourl: "https://github.com/RajaSrinivasan/repotrace.git"
      path: "repotrace"

    - repourl: "https://github.com/RajaSrinivasan/rollpwd.git"
      path: "rollpwd"
```

Project Groups The sample configuration lists two project groups: public and private. **public** projects are considered reference only and thus cloned and then detached from the HEAD. The repositories are considered immutable by this tool and thus push, tag and other operations will not be applied to the public projects.

Workarea Each project group requires a work area to be provided. This is the top level of the directory structure where the projects will be cloned. Each project's path specification is appended to the work area to determine the workarea for the project. For the project codex in the above example, the work area will be the folder `codex` under the work area **$HOME/grepo/pvt**. Please note that the path for each project has to be unique.

The work area specified is first translated using the environment variables.

Workspace directories should not fall within another git tree to avoid confusions.

Server The project group can have a default server specified. This will be used for each project unless overridden in the project. In the example all the private projects come from gitlab.com and accessed using the git protocol **git@gitlab.com:RajaSrinivasan/**.

Repository spec A project can provide one of **repo** which is combined with the Server of the project group. Alternatively a **repourl** can be specified which becomes the full specification of the repository. In the above example, for **srctrace** the entire url is provided whereas for **codex** the url is formed using the Server spec and the repo spec resulting in: **git@gitlab.com:RajaSrinivasan/codex.git**.

Reference Reference is a branch name or tag. The reference specification of the project group will be applied to each project unless overridden in the project. For a project, the reference could also be a commit id.

Build instruction A project can provide a build instruction. Upon completion of a clone or a pull, this command expected to be a shell command is executed. As an illustration, for the project codex in the example, a make command is specified.

3 Usage and examples

3.1 usage

Listing 2: Basic usage

```
$ bin/grepo --help

        grepo supports a project that comprises different repositories.

Usage:
  grepo [command]

Available Commands:
  diff        Diff from where we started
  help        Help about any command
  init        Initialize - setup the workspace
  pull        Pull for each repo
  push        Push for each repo
  status      Project Status
  version     Report the version of the application

Flags:
      --config string   config file. (default "Project.yaml")
  -h, --help            help for grepo
```

```
      --verbose          be verbose
  -v, --version          version for grepo

Use "grepo [command] --help" for more information about a command.
```

init The init command sets up the directory structure, clones the repository and performs the initial build of the repository. Optionally a branch name can be provided which creates the same branch on all the private repos.

pull After the initial setup, the pull command applies **git pull** to each of the projectlets followed by an optional build. The public repos can also be pull'ed and optionally a build performed.

tag Applies the tag to each (private only) repository. While pushing, the tags will be pushed as well.

diff For each private repository, this displays the difference from the original reference.

push For each private repository, this performs a commit and a push. The same commit message is applied to all the repositories. If a branch name had been specified with the init command, the push target is the newly created branch name. This utility applies a **commit -a** to each repo and thus can take care of the files known to git ie previously under git control. New files created should be added manually. Any tags created can also be pushed to the remote.

Listing 3: Project Initialization. Sample log

```
bin/grepo init --verbose --force
2020/05/30 06:56:39 Loading the config file Project.yaml
2020/05/30 06:56:39 Project group: Public
2020/05/30 06:56:39 Workspace /Users/rajasrinivasan/grepotest/pub
2020/05/30 06:56:39 Project group: Private
2020/05/30 06:56:39 Workspace /Users/rajasrinivasan/grepotest/pvt
...
2020/05/30 06:56:41 Executing /usr/bin/git clone git@gitlab.com:RajaSrinivasan/cod
2020/05/30 06:56:56 Cloning into '.'...
2020/05/30 06:56:56 Executing /usr/bin/git checkout v0.1.0-B
2020/05/30 06:56:56 Note: checking out 'v0.1.0-B'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 0a26e9b Deleted bin/app
...
2020/05/30 06:56:59 Created /Users/rajasrinivasan/grepotest/pvt/srctrace
2020/05/30 06:56:59 Executing /usr/bin/git clone https://github.com/RajaSrinivasan
2020/05/30 06:57:00 Cloning into '.'...
2020/05/30 06:57:00 Executing /usr/bin/git checkout feat_many_platforms
2020/05/30 06:57:00 Switched to a new branch 'feat_many_platforms'
Branch 'feat_many_platforms' set up to track remote branch 'feat_many_platforms' f
```

Listing 4: Example of status

```
$ bin/grepo status
2020/05/30 17:27:03 Status in /Users/rajasrinivasan/grepotest/pvt/random
2020/05/30 17:27:03 On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
2020/05/30 17:27:03 Status in /Users/rajasrinivasan/grepotest/pvt/exec
2020/05/30 17:27:03 On branch master
Your branch is up to date with 'origin/master'.
```

4 Implementation

Implementation in **go** : `https://github.com/RajaSrinivasan/grepo.git`